Serial No.

University of Bahrain
College of Information Technology
Department of Computer Science
Semester II, 2014-2015
ITCS104/ITCS112 (Computer Programming II)

Final Exam

Date: 09th June 2015 Time: 11:30-13:30

STUDENT NAME	ANSWER
STUDENT ID	
SECTION #	

QUESTION #	PART #	MARKS		Comments
Q1	PART A	12		
	PART B	16		
Q2	PART A	8		
	PART B	5		
	PART C	4		
Q3	PART A	6		
	PART B	9		
TOTAL		**60**		

NOTE: THERE ARE (9) PAGES IN THIS TEST

WRITE ONLY **ONE** SOLUTION FOR EACH QUESTION

Question 1 [Pointers, Struct, Classes, & Arrays]

Consider the following struct:

```
struct account{
    string userName; //  to represet a user name on a social network
    int numFollowers; //  to represent the number of followers for this user
    };
```

A class named **socialNetwork** should be designed to hold the following information:

- *netoworkName*: to represent the name of the social network, for example "Twitter", "Instagram", ….etc.
- *maxSize*: to represent the maximum number of accounts in this network.
- *numAccount*: to represent the current number of registered accounts in this network.
- *list*: a pointer of type **account**, representing a dynamic one-dimensional array of size *maxSize* to hold the accounts information.

The class should also include the following member functions:

1. A **Constructor** with default value parameters for *networkName* = "*none*", and *maxSize*=100. The function should create the dynamic array *list* of size *maxSize* and initialize *numAccount* to be zero.
2. A **Copy Constructor**.
3. A **Destructor**.
4. A set function to set the network name.
5. A function named **addAccount** that takes as parameter (**act**) of type **account.** The function should add **act** into **list** and increment **numAccounts.** The insertion should be done such that **list** should remain sorted in a descending order based on number of followers (i.e. the account with the highest number of followers should come first in **list**). The insertion should not be performed if there is no enough space in **list**.
6. A function named **print.** The function should display the network name and the list of accounts.
7. A function named **follow** that takes as parameter **userN** of type string. The function should increment the number of followers of the account that has the same name as **userN.**

Question 1 [Continue ..]

PART A [12 Points]

Declare the **socialNetwork** class [don't include the functions implementation]

```
class socialNetwork{
    private: //0.5 pt
      string networkName; // 1 pt
      int maxSize;  // 1 pt
      int numAccount;  // 1 pt
      account * list; // 1 pt
    public://0.5 pt
      socialNetwork(string n, int max=100);  // 1 pt
      socialNetwork(const socialNetwork&);  // 1 pt
      void setName (string s);  // 1 pt
      void addAccount(account act);  // 1 pt
      void print(); // 1 pt
      void follow (string act); // 1 pt
      ~socialNetwork( );  // 1 pt
};
```

Question 1 [Continue ..]

PART B [16 Points]

Using the class defined in Part (A), write the implementation of the following functions:

1) **Copy Constructor**

2) **Destructor**

```
socialNetwork::socialNetwork(const socialNetwork & obj){
   networkName = obj.networkName;
   maxSize= obj.maxSize;
   numAccount= obj.numAccount;
   list = new account[maxSize];
   for(int i=0; i<numAccount; i++)
      list[i] = obj.list[i];

}
```

```
socialNetwork::~socialNetwork(){
delete [] list;
list=NULL;

}
```

Question 1 [Continue]

PART B [Continue]:

Using the class defined in Part (A), write the implementation of the following functions:

3) **addAccount**

4) **follow**

```
void socialNetwork:: addAccount(account act){
  // insert the new account such that the account with highest # followers comes first
  if (numAccount==maxSize)
     cout<<"Fullllllll....";
  else{
  int index;

  for( index=0; index<numAccount; index++)
    if(list[index].numFollowers < act.numFollowers)
       break;

  //insert
  for(int i = numAccount; i>index; i--)
      list[i] = list[i-1];

  list[index]= act;
  numAccount++;
}

}
```

```
void socialNetwork:: follow (string act){
  for(int i=0; i<numAccount; i++)
    if(list[i].userName == act)
       list[i].numFollowers+=1;

}
```

Question 2 [Overloading & Templates]

Consider the following class to answer the questions in Part (A), (B) & (C).

```
class Arr{
 private:
    int items[20];
    int len;
public:
      Arr(int n= 4); // constructor to initialize len to n
      void print( ); // print the elements in items
      void readList( ); // prompt the user to enter the elements of items
      int CountItem( int x ); //search and count all occurrences of x in items
};
```

PART A [8 Points]

Rewrite Arr class as a template class to hold an array of any type. The new template class should also include a function to overload the not equal (!=) operator as a member function which should return true if the two objects don't have the same *len*, or unequal corresponding elements. Otherwise, the function should return false. **Note**: write the template class definition only; don't write the implementation of the functions.

```
template <class T> // 2 pts
class Arr{
 private:
    T * List;  // 2 pts
    int size;
public:
      Arr(int n= 4);
      void print( );
      void readList( );
      int CountItem( T item );  // 2 pts
      bool operator!=(const Arr&)const;  // 2 pts
};
```

Question 2 [Continue ..]

PART B [5 Points]

Write the implementation of the overloading function of the operator (!=) defined in the template class in Part (A).

```
template <class T>
bool Arr<T>::operator!=(const Arr& obj)const{  // 1pt
  if (size != obj.size)   // check size 1 pt
      return true;
  else{ // check elements = 3 pts
     for(int i =0; i <size; i++)
        if(List[i] != obj.List[i])
            return true;
  return false;
  }

}
```

PART C [4 points]

Using the template class defined in Part (A), write a main function to create two objects named *objA* and *objB* of type *Arr* to hold arrays of double numbers. Ask the user to enter the elements of the two objects, and then output a message to indicate whether or not the two objects are equal.

```
int main( ){
   Arr <double> objA ;      Arr <double> objB ;    // 1 pt

   objA.readList(  );      objB.readList( ); // 1pt

   if(objA !=objB) // 2pts
     cout<<"Not equal..";
   else
     cout<<"Equal...";


   return 0;}
```

Question 3 [Recursion]

PART A [6 Points]: Show the output of the following program:

```
int Recfunct(int list[], int x, int y){
    if (x == y)
       return list[x];

    else if(list[x]>list[y])
         {
           list[x]-=list[y];
           cout<< list[x]<< "\t" <<list[y]<<endl;
           return list[x] + Recfunct( list,x+1, y-1);
         }

    else
           {
           list[x]+=list[y];
           cout<< list[x]<< "\t" <<list[y]<<endl;
           return list[y] + Recfunct( list,x+1, y-1);

           }
}

int main( ) {
int list[5]={8,3,1,4,5};
cout<< Recfunct(list,0, 4);
return 0;
}
```

OUTPUT

3 5 // each 1 pt
7 4 // each 1 pt
8 // 2 pts

Question 3 [Continue]

PART B [9 Points]:

Write a **recursive** function that takes as parameters a string (*str*) and its size. The function should encrypt *str* such that the first character should be replaced by '*' and any character located at an even index should be replaced by '#'.

The function prototype is: *void encrypt (string & str, int size);*

For example:

If str= "ITCS-112", then the new value of str should be = "*T#S#1#2"

If str="MyExam!", then the new value of str should be = "*y#x#m#"

```
/*
If len== 0; replace by *  = 2 pts
If len is even; replace by #  = 3 pts
Call recursive function = 2 pts
Base case = 2pts

*/
void encode(string & str, int len){
 if (len>0)
     { len--;
       if (len==0)
          str[0]='*';
       else if (len%2==0)
          str[len]='#';

       encode(str, len-1);
     }
}
```